Using Synthetic Domain Theory to Prove Operational Properties of a Polymorphic Programming Language Based on Strictness

Simpson, A., & Rosolini, G. Using Synthetic Domain Theory to Prove Operational Properties of a Polymorphic Programming Language Based on Strictness.

Using Synthetic Domain Theory to Prove Operational Properties of a Polymorphic Programming Language Based on Strictness*

Giuseppe Rosolini
Dipartimento di Informatica e Scienze dell'Infomazione
Università di Genova, Italy

Alex Simpson
LFCS, School of Informatics
University of Edinburgh, UK

Abstract

We present a simple and workable axiomatization of domain theory within intuitionistic set theory, in which predomains are (special) sets, and domains are algebras for a simple equational theory. We use the axioms to construct a relationally parametric set-theoretic model for a compact but powerful polymorphic programming language, given by a novel extension of intuitionistic linear type theory based on strictness. By applying the model, we establish the fundamental operational properties of the language.

1. Introduction

The idea of *synthetic domain theory (SDT)* was proposed by Dana Scott around 1980. He suggested that some of the order-theoretic and topological complexities of domain theory might be avoided by simply taking domains to be special sets, and morphisms of domains to be arbitrary set-theoretic functions. Although such an idea is incompatible with ordinary *classical* set theory, Scott indicated that it should be consistent with *intuitionistic* set theory [20]. Subsequently, in a long line of research including [18, 9, 6, 21, 14, 22], a substantial theory has been developed, incorporating the full range of domain-theoretic constructions as set-theoretic constructions within intuitionistic set theory. To some extent, this work has fulfilled Scott's original hope of avoiding the order-theoretic and topological aspects of classical domain theory, yet only at the expense of introducing significant new difficulties of a logical and category-theoretic nature. Even for researchers in semantics, there has hitherto been little incentive to get to grips with the technical demands of SDT, as it has not been clear what the pay-off might be in terms of applications.

The two main goals of the present paper are: (i) to make synthetic domain theory accessible to a wider audience by presenting a notably simple axiomatization within intuitionistic set theory; and, more significantly, (ii) to demonstrate the applicability of this theory by using it to establish nontrivial operational properties of a compact yet powerful polymorphic programming language.

Concerning (i), our axiomatization differs from existing accounts in two main ways. In SDT, "domains" are traditionally defined as algebras for a lifting monad acting on a category of "predomains". We present an equivalent view that avoids the category-theoretic abstraction of working with algebras for a monad. Instead, in Sections 2 and 3, domains are defined as algebras (in the usual sense) for an equational theory. This simple reformulation turns out to be easy to work with concretely. Moreover, it brings lifting in line with the general algebraic approach to computational effects advocated in recent work of Plotkin and Power [13].

The second main difference from standard accounts of SDT is that, rather than favouring any particular one of the many synthetic notions of predomain, we merely axiomatize the basic properties of predomains and domains that are useful in practice. In Section 5, we show that our axioms suffice for building a model of polymorphism that is *relationally parametric* in the sense of Reynolds [15]. What is noteworthy about our approach to polymorphism is that it circumvents entirely the awkward indexing issues that usually accompany the technology of *small complete categories* [5, 16, 7] (see Remark 5.5).

Concerning (ii), our application is to establish operational properties of a polymorphic language (loosely) based on linear type theory. It was Plotkin who first realised the surprising power of combining linear type theory, polymorphism and recursion [12]. This observation influenced Bierman, Pitts and Russo to design a simple programming language, called `Lily`, based on only three type constructors: $\forall\alpha.\,\sigma$ for polymorphism, $!\sigma$ for "thunks" and $\sigma \multimap \tau$ for (linear) functions [2]. Following Plotkin's ideas, this compact language is capable of encoding a rich variety type constructs, including recursive types. The main contribution of [2] was to develop techniques based on operational semantics for reasoning about operational properties of `Lily`.

In this paper we show that synthetic domain theory of-

*Research supported by EPSRC Research Grant GR/S5594/01.

fers a denotational alternative to the operational methods of [2]. Our vehicle for demonstrating this is an extension of `Lily`, which we introduce in Section 4. Our language shares the same type structure as `Lily`. The difference is that we interpret $\sigma \multimap \tau$ more generously as a type of *strict* rather than *linear* functions. This leads to a more powerful language than `Lily` in the sense that our language assigns types to more programs. Our language is possibly of interest in its own right, as a novel and natural extension of linear type theory, and as a potential intermediate language for use in compilation as a target language for strictness analysis.

In Section 5, working within intuitionistic set theory, we build a relationally parametric model of our language. The existence of such a model is, in itself, already an advantage of our synthetic framework, as models of parametric polymorphism have not been forthcoming within the context of ordinary domain theory. Relational parametricity has many applications. For example, it can be used to justify the correctness of Plotkin's datatype encodings. Our aim in this paper, however, is to use the model to prove operational properties. In Section 6, we prove a *computational adequacy* result for our language showing that our model is sound for establishing operational equivalences determined by a strict (call-by-value) operational semantics. In Section 7, we prove a second computational adequacy result for a non-strict (call-by-name) variant of the operational semantics. The fact that computational adequacy holds for the same model with respect to both strict and non-strict operational semantics means that the operational equivalences induced by the two semantics coincide. We thus obtain a denotational proof that the *strictness theorem* of [2] extends to our language. In addition, we exploit the approximation relation defined in the proof of computational adequacy to establish an *operational extensionality* result, once again showing that properties for `Lily`, established in *op. cit.*, extend to our language. Finally, in Section 8, we give a brief justification that our methods, which are based on classically inconsistent axioms about sets, are nonetheless sound for establishing operational properties of programs.

This paper reveals synthetic domain theory to be a serious competitor to state-of-the-art techniques in operational semantics. Although our results exactly mirror those for `Lily` presented in [2], we establish them for a slightly richer language. While it seems likely that the syntactic methods of [2] should extend to our language,[1] such methods are themselves quite involved, and it is certainly worth showing that a complementary approach based on denotational semantics is possible.

Acknowledgements We thank Dana Scott and Phil Wadler for helpful suggestions.

[1] This is not completely trivial. For example, certain restrictions on occurrences of linear variables in `Lily` do not hold for our language.

2. Pointed Sets and Strict Functions

In traditional domain theory, a "domain" is a directed-complete partial order (a "predomain") that also has a least element (i.e. is "pointed"). Thus one has the equation:

$$\textit{domain} = \textit{pointed predomain}. \tag{1}$$

This equation is the model for our synthetic development. In the synthetic account, predomains are just special sets; we consider their properties in Section 3. First, in this section, we address the pointedness condition, which will be implemented using a notion of pointed set.

To assist the reader, it is convenient to review the classical notion of pointed set. Traditionally, a "pointed set" is a structure (A, a) where A is a set and $a \in A$ is any element, the "point". A "strict function" $f\colon (A, a) \to (B, b)$ is simply a function $f\colon A \to B$ such that $f(a) = b$. Equivalently, a pointed set is an algebra $(A, r_\perp)$ for a signature consisting of a single 0-ary operator (i.e. constant) $r_\perp$ and no equations, and a strict function is simply a homomorphism. Again equivalently, a pointed set is an algebra $(A, r_\perp, r_\top)$ for a signature consisting of two operators: a 0-ary operator $r_\perp$ and a unary operator $r_\top$ satisfying the equation

$$r_\top(x) = x. \tag{2}$$

Although seemingly a trivial reformulation, it is this latter presentation that adapts best to our purposes.

It is standard mathematical practice to work informally within classical set theory. As discussed in the introduction, the development of synthetic domain theory has to be carried out using a set theory based on intuitionistic logic. In this paper, we simply modify standard practice and work informally within intuitionistic set theory. To follow the development at an intuitive level, the reader is required merely to have some feeling for intuitionistic reasoning. Formally, the background set theory can be taken to be IZF [19].

Let $\mathbf{1}$ be the singleton set $\{\emptyset\}$. Its powerset $\mathcal{P}(\mathbf{1})$ is isomorphic to (or may be taken as the definition of) the set Ω of truth values, under the bijection mapping any subset $e \in \mathcal{P}(\mathbf{1})$ to the truth value of the proposition $\emptyset \in e$, and conversely mapping any truth value $p \in \Omega$ to the subsingleton set $\{\emptyset \mid p\}$ that contains $\emptyset$ iff p is true. (Equality on Ω is logical equivalence, i.e. $p = q$ iff $p \Leftrightarrow q$.) Of course $\{\top, \bot\} \subseteq \Omega$, where $\top$ is "true" and $\bot$ is "false". It will follow from later axioms that $\Omega \neq \{\top, \bot\}$, i.e. our logic is forced to be non-classical.

In the last of the accounts of classical pointed sets above, a pointed set was an algebra for a signature consisting of two operators $r_\perp$ and $r_\top$, i.e. for a family of operators $\{r_p\}_{p \in \{\bot, \top\}}$. Intuitionistically, we shall analogously consider families $\{r_p\}_{p \in \Sigma}$ indexed by a subset $\Sigma \subseteq \Omega$. Intuitively, Σ is to be understood as the set of truth values of

all propositions of the form "the execution of P terminates" where P ranges over all possible "computations". Rather than formalizing the notion of computation, we leave it abstract and instead axiomatize the properties we need of Σ.

Definition 2.1 (Dominance [18]) A subset $\Sigma \subseteq \Omega$ is a *dominance* if:

1. $\top \in \Sigma$, and
2. for all $p \in \Sigma$, $q \in \Omega$, if $p \to (q \in \Sigma)$ then $(p \wedge q) \in \Sigma$.

The set Ω and its subsets $\{\top, \bot\}$ and $\{\top\}$ are all easily seen to be dominances. Our first axiom asserts that our assumed set Σ of termination properties is also a dominance.

Axiom 1 The distinguished subset $\Sigma \subseteq \Omega$ is a dominance.

Remark 2.2 Axiom 1 does not imply that $\bot \in \Sigma$. Thus we are not yet asserting the existence of nonterminating computations. This will rather follow in Section 3 as a consequence of Axiom 4, which imposes the closure of computations under general recursion.

In the classical account of pointed sets, the arity of $r_\bot$ was 0 and that of $r_\top$ was 1. For a pointed set $(X, \{r_p\}_{p\in\Sigma})$, we generalize this by giving r_p the "arity" $\{\emptyset \mid p\}$ in the sense that r_p is a function from $X^{\{\emptyset \mid p\}}$ to X. In practice, it is convenient to work instead with an isomorphic description of $X^{\{\emptyset \mid p\}}$. Define

$$X^p = \{e \in \mathcal{P}(X) \mid (\forall x, y \in e,\, x = y) \wedge ((\exists x \in e) \Leftrightarrow p)\}\,,$$

where we write $(\exists x \in e)$ as shorthand for the proposition $(\exists x \in e,\, \top)$ stating that e is inhabited. It is easily shown that X^p is isomorphic to $X^{\{\emptyset \mid p\}}$.

Definition 2.3 (Pointed set) A *pointed set* is a structure $(X, \{r_p \colon X^p \to X\}_{p\in\Sigma})$ satisfying, for all $x \in X$, all $p, q \in \Sigma$ and $e \in X^{p\wedge q}$,

$$r_\top\{x\} = x\,, \tag{3}$$
$$r_{p\wedge q}(e) = r_p\{r_{p\wedge q}(e) \mid p\}\,. \tag{4}$$

Here, equation (3) is just equation (2) from before. Equation (4) is derivable for $\Sigma = \{\top, \bot\}$, and is thus redundant classically. The above equations are motivated by being just what is needed for Lemma 2.6 below to hold.

As in the classical case, pointed sets are algebras. Similarly, strict functions are just homomorphisms.

Definition 2.4 (Strict function) A *strict function* from a pointed set $(X, \{r_p^X\}_p)$ to another $(Y, \{r_p^Y\}_p)$ is a function $f \colon X \to Y$ satisfying, for all $e \in X^p$,

$$f(r_p^X(e)) = r_p^Y\{f(x) \mid x \in e\}. \tag{5}$$

When convenient, we shall leave the operator structure implicit when working with pointed sets, writing X rather than $(X, \{r_p\}_p)$. We write $X \multimap Y$ for the set of strict functions between pointed sets X, Y, and we write $X \cong^\circ Y$ to mean that X and Y are isomorphic via strict functions.

The category of pointed sets and strict functions enjoys all the usual properties of categories of algebra homomorphisms. For example, for any family $\{(Y_x, \{r_p^x\}_p)\}_{x\in X}$ of pointed sets, the product $(\prod_{x\in X} Y_x, \{r_p^{\prod_{x\in X} Y_x}\}_p)$ is pointed, where

$$r_p^{\prod_{x\in X} Y_x}(e) = \{\, r_p^x\{\pi_x \mid \pi \in e\}\,\}_{x\in X}.$$

As the set of functions $X \to Y$ is isomorphic to $\prod_{x\in X} Y$, it follows that, for any pointed set $(Y, \{r_p^Y\}_p)$, the function space $(X \to Y, \{r_p^{X\to Y}\}_p)$ is pointed, where

$$r_p^{X\to Y}(e) = (x \mapsto r_p^Y\{f(x) \mid f \in e\}).$$

Given a pointed set $(X, \{r_p^X\}_p)$, we say that a subset $Z \subseteq X$ is *subpointed* if the operators on X restrict to operators on Z, i.e. if, for all $p \in \Sigma$ and $e \in Z^p$, it holds that $r_p^X(e) \in Z$ (note that indeed $Z^p \subseteq X^p$). If X, Y are pointed and $f, g \colon X \to Y$ are strict, then the equalizer $\{x \in X \mid f(x) = g(x)\}$ is a subpointed subset of X.

Lemma 2.5 *If X, Y are pointed then $X \multimap Y$ is a subpointed subset of $X \to Y$.*

Lemma 2.6 (Lifting) *For any set X, the structure $(\mathsf{L}X, \{\mu_p\}_p)$ defined below is pointed.*

$$\mathsf{L}X = \bigcup_{p\in\Sigma} X^p \qquad\qquad \mu_p(E) = \bigcup E$$

Moreover, for any pointed set $(Y, \{r_p\}_p)$, and $f \colon X \to Y$, there exists a unique strict $f^* \colon (\mathsf{L}X, \{\mu_p\}_p) \to (Y, \{r_p\}_p)$ satisfying $f^*(\{x\}) = f(x)$ for all $x \in X$.

In other words, $\mathsf{L}X$ is the free pointed set generated by X. It also follows from the lemma that $\Sigma \cong \mathsf{L}\mathbf{1}$ is pointed.

3. Predomains and Domains

In this section we address the remaining components of equation (1). We separate the properties we require of predomains into two axioms. The first expresses closure under useful constructions on sets. The second is the key axiom for constructing models of polymorphism in Section 5.

Axiom 2 There is a class, $\mathbf{Predom}$, of special sets, called *predomains*, that satisfies:

1. If A is a predomain and $A \cong B$ then B is a predomain.

2. If $\{A_x\}_{x\in X}$ is a set-indexed family of predomains, then their product $\prod_{x\in X} A_x$ is a predomain.

3. For functions $f, g\colon A \to B$ between predomains, the equalizer $\{a \in A \mid f(a) = g(a)\}$ is a predomain.

4. The set of natural numbers $\mathbb{N}$ is a predomain.

5. If A is a predomain then so is its lifting $\mathsf{L}A$.

Axiom 3 There exists a set $\mathbf{P}$ of predomains such that, for every predomain A, there exists $B \in \mathbf{P}$ such that $B \cong A$.

Remark 3.1 Given Axiom 2, Axiom 3 is inconsistent with classical logic. It implies that the complete category of predomains is weakly equivalent to the small category whose objects are sets in $\mathbf{P}$ and whose morphisms are arbitrary functions. This small category is thus itself complete, but only in the weakest of the senses discussed in [16, 7].

Remark 3.2 By Freyd's adjoint functor theorem, Axiom 3 implies that the category of predomains is a full reflective subcategory of the category of sets, hence it is cocomplete.

Definition 3.3 (Domain) A domain is a pointed set $(A, \{r_p\}_p)$, where A is a predomain.

By the closure properties discussed after Definition 2.3, the category of strict functions between domains is complete. We write $\mathbf{Dom}$ for the class of domains. Define the set:

$$\mathbf{D} = \{(B, \{r_p\}_p) \mid B \in \mathbf{P} \text{ and } (B, \{r_p\}_p) \text{ is a pointed set}\}.$$

Lemma 3.4 *For every domain A there exists $D \in \mathbf{D}$ such that $D \cong^\circ A$.*

Remark 3.5 The lemma shows that the complete category of strict maps between domains is weakly equivalent to a small category. It follows that it is also cocomplete.

Axiom 4 For every domain $(A, \{r_p\}_p)$ there is a function $\mathrm{fix}_A\colon (A \to A) \to A$ satisfying:

1. (Fixed point property) For all $f\colon A \to A$ it holds that $f(\mathrm{fix}_A(f)) = \mathrm{fix}_A(f)$.

2. (Uniformity) For any domain $(B, \{r^B_p\}_p)$, and functions $f\colon A \to A$, $g\colon B \to B$ and strict $h\colon A \multimap B$, if $g \circ h = h \circ f$ then $\mathrm{fix}_B(g) = h(\mathrm{fix}_A(f))$.

Remark 3.6 Given Axiom 2, Axiom 4 is again inconsistent with classical logic as it implies the existence of nontrivial sets for which every endofunction has a fixed point.

Remark 3.7 It can be shown, see e.g. [23], that fix_A is uniquely determined by the property of uniformity. Moreover, by the dinaturality property of *loc. cit.*, fix_A does not depend on the algebra structure $\{r_p\}_p$.

4. A Polymorphic Language

In this section, we introduce our programming language, which is strongly influenced by the language `Lily` of [2]. It shares the same types as `Lily`, and has an equivalent language of raw terms. The important difference from `Lily` lies in the formulation of the typing rules. Our rules are more permissive. They correspond to interpreting $\sigma \multimap \tau$ as a type of *strict* as opposed to *linear* functions.

We use $\alpha, \beta, \ldots$ to range over type variables, and $\sigma, \tau, \ldots$ to range over types, which are given by:

$$\sigma \ ::= \ \alpha \mid \sigma \multimap \tau \mid !\,\tau \mid \forall\alpha.\,\sigma\,.$$

As usual, $\forall\alpha$ binds α, and we identify types up to renamings of bound variables. We write $\mathrm{ftv}(\sigma)$ for the set of free type variables of σ. if Θ is a finite set of type variables then we write $\sigma(\Theta)$ to mean that $\mathrm{ftv}(\sigma) \subseteq \Theta$.

We use $x, y, z, \ldots$ to range over term variables, and $s, t, \ldots$ to range over raw terms, which are given by:

$$\begin{aligned} t \ ::= x \mid \lambda x\!:\!\sigma.\ t \mid s(t) \mid !\,t \mid \mathrm{let}\ !x = s\ \mathrm{in}\ t \mid \\ \Lambda\alpha.\ t \mid t(\sigma) \mid \mathrm{rec}\, x\!:\!\sigma.\ t\,. \end{aligned}$$

Here x is bound in $\lambda x : \sigma.\ t$ and in $\mathrm{rec}\, x : \sigma.\ t$, and occurrences of x in t are bound in $\mathrm{let}\ !x = s\ \mathrm{in}\ t$. We again identify terms up to renamings of bound variables, and we write $\mathrm{fv}(t)$ for the set of free variables in a term t.

The typing rules make use of *labelled contexts*, which we define first and then motivate below. As usual, a *context* is a function Γ mapping a finite set of variables, $\mathrm{dom}(\Gamma)$, to types. A *labelling* of Γ is a function from $\mathrm{dom}(\Gamma)$ to $\{0, 1\}$. The structure of typing judgments is

$$\Gamma \mid \delta \ \vdash_\Theta \ t : \sigma,$$

where $\Gamma \mid \delta$ is a *labelled contex* consisting of a type assignment Γ together with a labelling δ of Γ. A typing judgment is *well formed* if: $\mathrm{ftv}(\Gamma, \sigma) \subseteq \Theta$ and $\mathrm{fv}(t) \subseteq \mathrm{dom}(\Gamma)$.

Remark 4.1 Even though it refers to concepts yet to be defined, it is helpful to give the intuitive motivation behind the labelling, If $\Gamma \mid \delta \ \vdash_\Theta \ t : \sigma$ is derivable and $x \in \mathrm{dom}(\Gamma)$ then $\delta(x) = 1$ implies that the term t is *strict in* x in the following sense. Given closed terms $\{s_y\}_{y\in\mathrm{dom}(\Gamma)}$, of appropriate types, then, in any evaluation of a term of the form $E[t[s_y/y]_{y\in\mathrm{dom}(\Gamma)}]$, where $E[\cdot]$ is a *ground evaluation context* expecting an argument of type σ, it must hold that at least one occurrence of s_x is evaluated. If instead $\delta(x) = 0$ then no guarantees are available about the operational behaviour of t with respect to terms substituted for x.

The typing rules are given in Figure 1, and are to be read as applying only when the premises and conclusion are all

$$\frac{}{\Gamma \mid \mathbf{0},\ x :_1 \sigma \ \vdash_\Theta\ x : \sigma}\ (\text{var}) \qquad \frac{\Gamma \mid \delta,\ x :_1 \sigma \ \vdash_\Theta\ t : \tau}{\Gamma \mid \delta \ \vdash_\Theta\ \lambda x{:}\sigma.\, t : \sigma \multimap \tau}\ (\text{lam}) \qquad \frac{\Gamma \mid \delta \ \vdash_\Theta\ s : \sigma \multimap \tau \qquad \Gamma \mid \delta' \ \vdash_\Theta\ t : \sigma}{\Gamma \mid \delta \vee \delta' \ \vdash_\Theta\ s(t) : \tau}\ (\text{app})$$

$$\frac{\Gamma \mid \delta \ \vdash_\Theta\ t : \sigma}{\Gamma \mid \mathbf{0} \ \vdash_\Theta\ !t : !\sigma}\ (\text{bang}) \qquad \frac{\Gamma \mid \delta \ \vdash_\Theta\ s : !\sigma \qquad \Gamma \mid \delta',\ x :_{\_} \sigma \ \vdash_\Theta\ t : \tau}{\Gamma \mid \delta \vee \delta' \ \vdash_\Theta\ \text{let}\, !x = s \,\text{in}\, t : \tau}\ (\text{let})$$

$$\frac{\Gamma \mid \delta \ \vdash_{\Theta,\alpha}\ t : \sigma}{\Gamma \mid \delta \ \vdash_\Theta\ \Lambda\alpha.\, t : \forall\alpha.\, \sigma}\ (\text{Lam}) \qquad \frac{\Gamma \mid \delta \ \vdash_\Theta\ t : \forall\alpha.\, \sigma}{\Gamma \mid \delta \ \vdash_\Theta\ t(\tau) : \sigma[\tau/\alpha]}\ (\text{App}) \qquad \frac{\Gamma \mid \delta,\ x :_{\_} \sigma \ \vdash_\Theta\ t : \sigma}{\Gamma \mid \delta \ \vdash_\Theta\ \text{rec}\, x{:}\sigma.\, t : \sigma}\ (\text{rec})$$

Figure 1. Typing Rules

$$\frac{}{\lambda x{:}\sigma.\, t \Downarrow \lambda x{:}\sigma.\, t} \qquad \frac{s \Downarrow^s \lambda x{:}\sigma.\, s' \qquad t \Downarrow^s v' \qquad s'[v'/x] \Downarrow^s v}{s(t) \Downarrow^s v} \qquad \frac{s \Downarrow^n \lambda x{:}\sigma.\, s' \qquad s'[t/x] \Downarrow^n v}{s(t) \Downarrow^n v}$$

$$\frac{}{!t \Downarrow !t} \qquad \frac{s \Downarrow !\,s' \qquad t[s'/x] \Downarrow v}{\text{let}\, !x = s \,\text{in}\, t \Downarrow v} \qquad \frac{}{\Lambda\alpha.\, t \Downarrow \Lambda\alpha.\, t} \qquad \frac{t \Downarrow \Lambda\alpha.\, t' \qquad t'[\sigma/\alpha] \Downarrow v}{t(\sigma) \Downarrow v} \qquad \frac{t[\text{rec}\, x{:}\sigma.\, t\,/\,x] \Downarrow v}{\text{rec}\, x{:}\sigma.\, t \Downarrow v}$$

Figure 2. Evaluation relations

well formed. The rules make use of the following notation. We write Θ, α to mean $\Theta \cup \{\alpha\}$, where we assume that $\alpha \notin \Theta$. Under the assumption that $x \notin \mathrm{dom}(\Gamma)$, we write $\Gamma \mid \delta,\ x :_0 \sigma$ for the context consisting of the type assignment $\Gamma,\ x : \sigma$ and the labelling $\delta[0/x]$. Similarly, we write $\Gamma \mid \delta,\ x :_1 \sigma$ for the context consisting of the type assignment $\Gamma,\ x : \sigma$ and the labelling $\delta[1/x]$. The notation $\Gamma \mid \delta,\ x :_{\_} \sigma$ is used to represent either of the contexts $\Gamma \mid \delta,\ x :_0 \sigma$ and $\Gamma \mid \delta,\ x :_1 \sigma$. We also make use of the join semilattice operations on labellings of Γ under the pointwise ordering (where the ordering on $\{0, 1\}$ is $0 \leq 1$). We write: $\mathbf{0}$ for the least labelling of Γ (the everywhere 0 labelling); and $\delta \vee \delta'$ for the join of δ and δ'.

Remark 4.2 Our language relates to the language `Lily` of [2] as follows. Given a term $\Gamma; \Delta \ \vdash_\Theta\ t\ :\ \sigma$ of `Lily`, where $\Gamma; \Delta$ is a "dual" intuitionistic/linear context, there exists a (unique) labelling δ of the concatenated context Γ, Δ such that $\delta(x) = 1$ for all $x \in \mathrm{dom}(\Delta)$ and $\Gamma, \Delta \mid \delta \ \vdash_\Theta\ t : \sigma$.[2] Thus every term typable in `Lily` has a type in our language. The converse fails. In fact, our language is equivalent to extending `Lily` with a contraction principle that contracts $\Gamma, x : \sigma; y : \sigma, \Delta \ \vdash_\Theta\ t : \sigma$ to $\Gamma; y : \sigma, \Delta \ \vdash_\Theta\ t[y/x] : \sigma$. Such contraction across the intuitionistic/linear divide is stronger than the contraction (within the linear context itself) of relevant logic. We believe that our form of contraction is the natural one for dealing with strictness issues. Given that strictness (as opposed to linearity) is often the important issue in determining an efficient evaluation strategy, it is plausible that languages based on our contraction principle might serve as useful intermediate languages in compilation, e.g. as target languages for strictness analysis.

The reason for formulating the typing rules as we have (rather than, e.g., using an explicit contraction rule) is so that the rules are syntax directed. In fact, labellings as well as types are uniquely determined by terms and contexts.

Lemma 4.3 *If both* $\Gamma \mid \delta \ \vdash_\Theta\ t : \sigma$ *and* $\Gamma \mid \delta' \ \vdash_\Theta\ t : \sigma'$ *then* $\delta = \delta'$ *and* $\sigma = \sigma'$.

As in [2], we give two operational semantics to our language. In both, *values* are closed terms of the form:

$$v\ ::=\ \lambda x{:}\sigma.\, t \mid !\,t \mid \Lambda\alpha.\, t\,.$$

Figure 2 defines evaluation relations $t \Downarrow^s v$ and $t \Downarrow^n v$ between closed terms t and values v. The strict (or call-by-value) relation $t \Downarrow^s v$ is inductively defined by the specific $\Downarrow^s$ rule for application together with all rules written using the neutral $\Downarrow$ notation. Similarly, the non-strict (or call-by-name) relation $t \Downarrow^n v$ is defined by the $\Downarrow^n$ application rule

[2] We are ignoring the inessential syntactic difference between the recursively defined thunks of [2] and our explicit recursion operator, as the two are interdefinable, see *op. cit.*

together with the neutral rules. We write $t \Downarrow^s$ (resp. $t \Downarrow^n$) to mean that there exists v such that $t \Downarrow^s v$ (resp. $t \Downarrow^n v$). We write $t \simeq^s t'$ for strict Kleene equality: $t \Downarrow^s v$ iff $t' \Downarrow^s v$.

Both semantics are easily seen to be deterministic: if $t \Downarrow^s v$ and $t \Downarrow^s v'$ then $v = v'$ (and similarly for $\Downarrow^n$). Also, both are type sound: if $t : \sigma$ and $t \Downarrow^s v$ then $v : \sigma$ (and similarly for $\Downarrow^n$), where we write $t : \sigma$ to mean that t is a closed term of closed type σ.

As in [2], we define contextual (approximation and) equivalence using termination at types of the form $!\sigma$ as observations. In natural extensions of the language with additional type primitives, this turns out to be equivalent to observing termination at ground types only. This choice has many benefits, including: inducing extensionality properties for function and universal types, and the operational correctness of Plotkin's polymorphic encodings of type constructs. See [2] for a thorough discussion of these issues for the language `Lily`.

For closed σ, a *ground σ-context* is a term of the form $x\colon\_\sigma \vdash C : !\tau$. We usually denote such a context by $C[\cdot]$, and write $C[t]$ for $C[t/x]$, where $t : \sigma$.

Definition 4.4 (Contextual approximation/equivalence) For $t, t' : \sigma$, we write $t \sqsubseteq_{\mathrm{gnd}} t'$ to mean that, for all ground σ-contexts it holds that $C[t] \Downarrow^s$ implies $C[t'] \Downarrow^s$. We write $t \equiv_{\mathrm{gnd}} t'$ to mean that both $t \sqsubseteq_{\mathrm{gnd}} t'$ and $t' \sqsubseteq_{\mathrm{gnd}} t$ hold.

We do not introduce a non-strict variant of contextual approximation, because it follows from the theorem below that this coincides with the strict version above.

Theorem 4.5 (Strictness) *If $t : !\sigma$ then $t \Downarrow^s$ iff $t \Downarrow^n$.*

We shall prove this theorem in Section 7. Until we have the proof, the reader must keep in mind that $\sqsubseteq_{\mathrm{gnd}}$ and $\equiv_{\mathrm{gnd}}$ are defined using the strict evaluation relation.

Remark 4.6 Because `Lily` is included in our language, the strictness theorem for `Lily` [2, Theorem 2.3] is a consequence of Theorem 4.5 above.

Contextual approximation and equivalence can be hard to reason with directly. To address this, we introduce a complementary applicative (bi)simulation relation, which is more amenable to certain forms of argument, and we prove *operational extensionality*: the coincidence of ground contextual approximation and applicative simulation.

Definition 4.7 (Strict applicative simulation) The relation $\sqsubseteq^s_{\mathrm{app}}$ is the largest relation between closed terms of identical closed type satisfying:

1. if $t \sqsubseteq^s_{\mathrm{app}} t' \colon \sigma \multimap \tau$ then, for all values $v \colon \sigma$, it holds that $t(v) \sqsubseteq^s_{\mathrm{app}} t'(v) \colon \tau$;
2. if $t \sqsubseteq^s_{\mathrm{app}} t' \colon !\sigma$ then $t \Downarrow^s !s$ implies $t' \Downarrow^s !s'$ where $s \sqsubseteq^s_{\mathrm{app}} s' \colon \sigma$; and
3. if $t \sqsubseteq^s_{\mathrm{app}} t' \colon \forall\alpha.\,\sigma$ then, for all closed τ, it holds that $t(\tau) \sqsubseteq^s_{\mathrm{app}} t'(\tau) \colon \sigma[\tau/\alpha]$.

Strict applicative simulation has an alternative, more elementary, characterization. A *ground evaluation context* is a ground context generated by the grammar below:

$$E[\cdot] ::= [\cdot] \mid E[(\cdot)(V)] \mid E[(\cdot)(\tau)] \mid \mathrm{let}\ !x = (\cdot)\ \mathrm{in}\ E[x]\,.$$

Proposition 4.8 *If $t, t' : \sigma$ then $t \sqsubseteq^s_{\mathrm{app}} t'$ iff $E[t] \Downarrow^s$ implies $E[t'] \Downarrow^s$ for all ground evaluation σ-contexts $E[\cdot]$.*

Theorem 4.9 (Operational extensionality) *If $t, t' : \sigma$ then $t \sqsubseteq_{\mathrm{gnd}} t'$ if and only if $t \sqsubseteq^s_{\mathrm{app}} t'$.*

Again, we prove this theorem in Section 7.

5. A Relationally Parametric Model

In this section, we construct a relationally parametric model of our language. To do this, we give two interpretations of types: as domains, and as relations.

For D a *domain*, a *subdomain* of D is any subpointed subset $D' \subseteq D$ that is also a predomain. If D, E are domains then an *admissible relation* between D and E is a subdomain of the domain $D \times E$. We write $\mathcal{R}(D, E)$ for the set of all admissible relations.

Lemma 5.1 *If D, E are domains and $f \colon D \multimap E$ then*

$$\mathrm{graph}(f) = \{(d, e) \mid f(d) = e\}$$

is an admissible relation between D and E.

In particular, for any domain D, the diagonal relation $\Delta_D = \{(x, x) \mid x \in D\} = \mathrm{graph}(\mathsf{id}_D)$ is admissible.

Given a set of type variables Θ, a Θ-*environment* is a function $\gamma \colon \Theta \to \mathbf{Dom}$, and a *relational* Θ-*environment* is a tuple $\gamma^{\mathcal{R}} = (\gamma_1, \gamma_2, \gamma_R)$, where γ_1, γ_2 are Θ-environments, and $\gamma_R \in \prod_{\alpha\in\Theta} \mathcal{R}(\gamma_1(\alpha), \gamma_2(\alpha))$. For each type $\sigma(\Theta)$ and Θ-environment γ, we define a domain $[\![\sigma]\!]_\gamma$ and, for each relational Θ-environment $\gamma^{\mathcal{R}}$, we define an admissible relation $[\![\sigma]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}} \in \mathcal{R}([\![\sigma]\!]_{\gamma_1}, [\![\sigma]\!]_{\gamma_2})$. The interdependent definitions are given in Figure 3. In them, we write $\Delta^{\mathcal{R}}_\gamma$ for the relational Θ-environment $(\gamma, \gamma, \alpha \mapsto \Delta_{\gamma(\alpha)})$ determined by a Θ-environment γ. That these definitions are good can be shown using Axiom 2.

The relational interpretation of types implies that open types act functorially on the (large) groupoid of strict isomorphisms between domains, cf. [10, 3].

Lemma 5.2 (Groupoid action) *If $\sigma(\Theta, \alpha)$ then, for any Θ-environment γ, domains D_1, D_2 and isomorphism $i \colon D_1 \multimap D_2$, there exists a unique isomorphism $\mathrm{gpd}(\sigma, \gamma, i) \colon [\![\sigma]\!]_{\gamma[D_1/\alpha]} \multimap [\![\sigma]\!]_{\gamma[D_2/\alpha]}$ such that*

$$[\![\sigma]\!]^{\mathcal{R}}_{\Delta^{\mathcal{R}}_\gamma[(D_1, D_2, \mathrm{graph}(i))/\alpha]} = \mathrm{graph}(\mathrm{gpd}(\sigma, \gamma, i)).$$

Moreover, the mapping $i \mapsto \mathrm{gpd}(\sigma, \gamma, i)$ is functorial.

$$
\begin{aligned}
[\![\alpha]\!]_\gamma &= \gamma(\alpha)\\
[\![\sigma \multimap \tau]\!]_\gamma &= [\![\sigma]\!]_\gamma \multimap [\![\tau]\!]_\gamma\\
[\![!\,\sigma]\!]_\gamma &= \mathsf{L}[\![\sigma]\!]_\gamma\\
[\![\forall\alpha.\,\sigma]\!]_\gamma &= \{\pi \in \prod_{D\in\mathbf{D}} [\![\sigma]\!]_{\gamma[D/\alpha]} \mid \forall D_1, D_2 \in \mathbf{D},\, \forall R \in \mathcal{R}(D_1, D_2),\, [\![\sigma]\!]^{\mathcal{R}}_{\Delta_\gamma[R/\alpha]}(\pi_{D_1}, \pi_{D_2})\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\alpha]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(d_1, d_2) &\Leftrightarrow \gamma_R(\alpha)(d_1, d_2)\\
[\![\sigma \multimap \tau]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(f_1, f_2) &\Leftrightarrow \forall d_1 \in [\![\sigma]\!]_{\gamma_1}, d_2 \in [\![\sigma]\!]_{\gamma_2},\, [\![\sigma]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(d_1, d_2) \;\rightarrow\; [\![\tau]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(f_1(d_1), f_2(d_2))\\
[\![!\,\sigma]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(e_1, e_2) &\Leftrightarrow (\forall d_1 \in [\![\sigma]\!]_{\gamma_1},\, e_1 = \{d_1\} \;\rightarrow\; \exists d_2 \in [\![e_2]\!]_{\gamma_2},\, [\![\sigma]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(d_1, d_2)\,) \wedge\\
&\qquad (\forall d_2 \in [\![\sigma]\!]_{\gamma_2},\, e_2 = \{d_2\} \;\rightarrow\; \exists d_1 \in [\![e_1]\!]_{\gamma_1},\, [\![\sigma]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(d_1, d_2)\,)\\
[\![\forall\alpha.\,\sigma]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}}(\pi_1, \pi_2) &\Leftrightarrow \forall D_1, D_2 \in \mathbf{D}, \forall R \in \mathcal{R}(D_1, D_2),\, [\![\sigma]\!]^{\mathcal{R}}_{\gamma^{\mathcal{R}}[R/\alpha]}(\pi_{D_1}, \pi_{D_2})\,.
\end{aligned}
$$

Figure 3. Interpretation of Types

$$
\begin{aligned}
[\![x]\!]_{\gamma,\rho} &= \rho(x)\\
[\![\lambda x\!:\!\sigma.\; t]\!]_{\gamma,\rho} &= (d\colon [\![\sigma]\!]_\gamma \mapsto [\![t]\!]_{\gamma,\rho[d/x]})\\
[\![s(t)]\!]_{\gamma,\rho} &= [\![s]\!]_{\gamma,\rho}([\![t]\!]_{\gamma,\rho})\\
[\![!\,t]\!]_{\gamma,\rho} &= \{[\![t]\!]_{\gamma,\rho}\}\\
[\![\mathrm{let}\; !x = s \;\mathrm{in}\; t]\!]_{\gamma,\rho} &= (d\colon [\![\sigma]\!]_\gamma \mapsto [\![t]\!]_{\gamma,\rho})^*\,([\![s]\!]_{\gamma,\rho})\\
[\![\Lambda\alpha.\; t]\!]_{\gamma,\rho} &= \{[\![t]\!]_{\gamma[D/\alpha],\rho}\}_{D\in\mathbf{D}}\\
[\![t(\tau)]\!]_{\gamma,\rho} &= \mathrm{gpd}(\sigma,\gamma,i)(([\![t]\!]_{\gamma,\rho})_D) \text{ where } D \in \mathbf{D} \text{ and } i\colon D \multimap [\![\tau]\!]_\gamma \text{ is iso}\\
[\![\mathrm{rec}\, x\!:\!\sigma.\; t]\!]_{\gamma,\rho} &= \mathrm{fix}(d\colon [\![\sigma]\!]_\gamma \mapsto [\![t]\!]_{\gamma,\rho[d/x]})
\end{aligned}
$$

Figure 4. Interpretation of Terms

Corollary 5.3 (Identity extension) *For any type $\sigma(\Theta)$ and Θ-environment γ, $[\![\sigma]\!]^{\mathcal{R}}_{\Delta^{\mathcal{R}}_\gamma} = \Delta_{[\![\sigma]\!]_\gamma}$.*

Identity extension implies that all elements in the interpretation of polymorphic types $\forall\alpha.\,\sigma$ are relationally parametric.

Next, we define the interpretation of terms. Given a Θ-context Γ and a Θ-environment γ, then a Γ-γ-environment is any $\rho \in \prod_{x\in\mathrm{dom}(\Gamma)}[\![\Gamma(x)]\!]_\gamma$. A term $\Gamma \mid \delta \;\vdash_\Theta\; t : \sigma$ is interpreted as an element $[\![t]\!]_{\gamma,\rho} \in [\![\sigma]\!]_\gamma$, relative to any Θ-environment γ and Γ-γ-environment ρ. The definition of $[\![t]\!]_{\gamma,\rho}$ is given in Figure 4, where terms and subterms are assumed to have the types given in Figure 1.

Lemma 5.4 (Interpretation of terms) *If $\Gamma \mid \delta \;\vdash_\Theta\; t : \sigma$:*

1. *(Well-definedness) For any Θ-environment γ and Γ-γ-environment ρ, the value $[\![t]\!]_{\gamma,\rho} \in [\![\sigma]\!]_\gamma$ is well-defined.*
2. *(Strictness) If $x : \tau \in \Gamma$ and $\delta(x) = 1$ then for any Θ-environment γ and Γ-γ-environment ρ, the function $d \mapsto [\![t]\!]_{\gamma,\rho[d/x]}\colon [\![\tau]\!]_\gamma \to [\![\sigma]\!]_\gamma$ is strict.*
3. *(Relational parametricity) For any relational Θ-environment $\gamma^{\mathcal{R}}$, any Γ-γ_1-environment ρ_1 and Γ-γ_2-environment ρ_2, define*

   $$[\![\Gamma]\!]_{\gamma_R}(\rho_1, \rho_2) \Leftrightarrow \forall x \in \mathrm{dom}(\Gamma),\, [\![\Gamma(x)]\!]_{\gamma_R}(\rho_1(x), \rho_2(x)).$$

   Then $[\![\Gamma]\!]_{\gamma_R}(\rho_1, \rho_2)$ implies $[\![\sigma]\!]_{\gamma_R}([\![t]\!]_{\gamma_1,\rho_1}, [\![t]\!]_{\gamma_2,\rho_2})$.

Remark 5.5 The form of completeness that holds for the small category $\mathbf{D}$ (see Remark 3.1) is too weak for interpreting polymorphism in $\mathbf{D}$, see [16]. We side-step the problem by, instead, using the non-small (but properly complete) category $\mathbf{Dom}$ for the interpretation. The one difficulty that arises lies in interpreting type specialization. For this, the invariance of the interpretation of universal types under groupoid action, cf. [10, 3], which follows from parametricity, is crucial to showing that the definition of $[\![t(\tau)]\!]_{\gamma,\rho}$ given in Figure 4 is independent of the choice of i and D.

Remark 5.6 Birkedal, Mogelberg and Petersen[3] have been

[3] Private communication.

$$
\begin{aligned}
d \preceq^{\alpha}_{\zeta} t \;&\Leftrightarrow d \preceq_{\alpha} t \\
f \preceq^{\sigma_1 \multimap \sigma_2}_{\zeta} s \;&\Leftrightarrow f\!\downarrow \rightarrow\; (s \Downarrow^s \lambda x{:}\sigma_1.\, s' \text{ and } \forall d \in [\![\sigma_1]\!]_\gamma,\, \forall t{:}\,\sigma_1[\vec{\tau}/\Theta],\; d \preceq^{\sigma_1}_{\zeta} t \;\rightarrow\; f(d) \preceq^{\sigma_2}_{\zeta} s'[t/x]) \\
e \preceq^{!\sigma}_{\zeta} t \;&\Leftrightarrow \forall d \in [\![\sigma]\!]_\gamma,\, e = \{d\} \;\rightarrow\; (\exists t' : \sigma,\; t \Downarrow^s !\, t' \text{ and } d \preceq^{\sigma}_{\zeta} t') \\
\pi \preceq^{\forall\alpha.\,\sigma}_{\zeta} t \;&\Leftrightarrow \pi\!\downarrow \rightarrow\; (t \Downarrow^s \Lambda\alpha.\, t' \text{ and } \forall D \in \mathbf{D},\, \forall \tau,\, \forall \preceq \in \mathcal{A}^s(D,\tau),\; \pi_D \preceq^{\sigma}_{\zeta[(D,\tau,\preceq)/\alpha]} t'[\tau/\alpha])
\end{aligned}
$$

Figure 5. Interpretation of Types as Strict Approximation Relations

studying a domain-theoretic version of the "parametric completion" process of [17]. It would be interesting to compare the category-theoretic model they obtain with the environment model constructed here.

Having constructed a relationally parametric model, we could, at this point, go on to prove useful consequences of parametricity. One of the most important consequences is the correctness of Plotkin's impredicative encodings of the domain-theoretic type constructors (see [2] for details of the encodings, and for a proof of correctness for coproducts). However, for lack of space, and because the techniques needed for such applications of parametricity are known, we omit reworking the expected verifications in our setting. Instead, for the remainder of the paper, we concentrate on showing how our model can be used to prove operational properties of our language. In particular, we obtain denotational proofs of Theorems 4.5 and 4.9.

6. Computational Adequacy

In order to use our model to prove operational properties, it is necessary to prove *computational adequacy*.

For x, y in a set X, we write $x \sqsubseteq_\Sigma y$ to mean that, for all $t\colon X \to \Sigma$, it holds that $t(x)$ implies $t(y)$. The $\sqsubseteq_\Sigma$ relation is a preorder (but not necessarily a partial order).

Theorem 6.1 (Computational adequacy) *The following equivalent properties all hold.*

1. *If $t : !\,\sigma$ then $t \Downarrow^s$ if and only if $\exists d \in [\![\sigma]\!],\, [\![t]\!] = \{d\}$.*
2. *If $s, t : \sigma$ then $[\![s]\!] \sqsubseteq_\Sigma [\![t]\!]$ implies $s \sqsubseteq_{\mathrm{gnd}} t$.*
3. *If $s, t : \sigma$ then $[\![s]\!] = [\![t]\!]$ implies $s \equiv_{\mathrm{gnd}} t$.*

We prove statement 1. The left-to-right implication is easily shown by proving that $t \Downarrow^s v$ implies $[\![t]\!] = [\![v]\!]$, by induction on the evaluation relation. The converse implication is proved by constructing an "approximation relation" between syntax and semantics. The construction is reminiscent of Girard's method of proving strong normalization for the polymorphic λ-calculus, see e.g. [4]. A similar approach to computational adequacy for a polymorphic language was previously taken by Amadio [1], who worked concretely with PER models. Our language, with its treatment of strictness, is more refined than Amadio's (e.g., ours supports Plotkin's encodings of datatypes), and our proof works within the purely axiomatic setting of this paper.

The crucial point in formulating a usable notion of approximation relation is the identification of a suitable notion of "definedness" for an arbitrary domain D. For $d \in D$, we write $d\!\downarrow$ to mean that there exists a strict function $t\colon D \multimap \Sigma$ such that $t(d)$.

For a domain D and closed type σ, a *strict approximation relation* between D and σ is a relation $\preceq$ between elements of D and closed terms of type σ satisfying:

(sa1) $\{d \mid d \preceq t\}$ is a subdomain of D.

(sa2) If $d\!\downarrow$ implies $d \preceq t$ then $d \preceq t$.

(sa3) If $d \preceq t$ and $t \simeq^s t'$ then $d \preceq t'$.

(sa4) If $d \preceq t$ and $d\!\downarrow$ then $t \Downarrow^s$.

We write $\mathcal{A}^s(D, \sigma)$ for the set of all strict approximation relations between D and σ.

Given a set of type variables Θ, a Θ-*substitution* is a family $\vec{\tau} = \{\tau_\alpha\}_{\alpha\in\Theta}$ of closed types. For a type $\sigma(\Theta)$, we write $\sigma[\vec{\tau}/\Theta]$ for the evident closed type resulting from the substitution. A *strict approximation* Θ-*environment* is a triple $\zeta = (\gamma, \vec{\tau}, \{\preceq_\alpha\}_{\alpha\in\Theta})$, where γ is a Θ-environment, $\vec{\tau}$ is a Θ-substitution and $\preceq_\alpha \;\in\; \mathcal{A}^s(\gamma(\alpha), \tau_\alpha)$. For any type $\sigma(\Theta)$ and strict approximation Θ-environment $\zeta = (\gamma, \vec{\tau}, \{\preceq_\alpha\}_\alpha)$, Figure 5 defines a strict approximation relation $\preceq^\sigma_\zeta \;\in\; \mathcal{A}^s([\![\sigma]\!]_\gamma, \sigma[\vec{\tau}/\Theta])$

Remark 6.2 It takes quite some work to verify that $\preceq^\sigma_\zeta$, is indeed a strict approximation relation. In fact, this is the first place in the paper that property 4 of Axiom 2 is used.

Remark 6.3 To the reader acquainted with proofs of computational adequacy, the use of substitutions $s'[t/x]$ for arbitrary terms t rather than just values, in the definition of $\preceq^{\sigma_1\multimap\sigma_2}_\zeta$, may appear to conflict with the strict operational semantics. However, for our language, the proof of computational adequacy is insensitive to this issue: one could indeed restrict to values, but there is no need to do so.

Given a Θ-context Γ and a Θ-substitution $\vec{\tau}$, a Γ-$\vec{\tau}$-substitution is a family $\vec{t} = \{t_x\colon \Gamma(x)[\vec{\tau}/\Theta]\}_{x\in\mathrm{dom}(\Gamma)}$.

$$
\begin{aligned}
d \precsim_\zeta^\alpha t &\Leftrightarrow d \precsim_\alpha t \\
f \precsim_\zeta^{\sigma_1 \multimap \sigma_2} s &\Leftrightarrow \forall d \in [\![\sigma_1]\!]_\gamma,\, \forall t\colon \sigma_1[\vec{\tau}/\Theta],\ d \precsim_\zeta^{\sigma_1} t \ \rightarrow\ f(d) \precsim_\zeta^{\sigma_2} s(t) \\
e \precsim_\zeta^{!\sigma} t &\Leftrightarrow \forall d \in [\![\sigma]\!]_\gamma,\, e = \{d\} \ \rightarrow\ (\exists t' : \sigma,\ t \Downarrow^n !\, t' \text{ and } d \precsim_\zeta^\sigma t') \\
\pi \precsim_\zeta^{\forall\alpha.\,\sigma} t &\Leftrightarrow \forall D \in \mathbf{D},\, \forall \tau,\, \forall \precsim \in \mathcal{A}^n(D,\tau),\ \pi_D \precsim_{\zeta[(D,\tau,\precsim)/\alpha]}^{\sigma} t(\tau).
\end{aligned}
$$

Figure 6. Interpretation of Types as Non-strict Approximation Relations

Lemma 6.4 *Suppose* $\Gamma \mid \delta \ \vdash_\Theta \ s : \sigma$. *For any strict approximation* Θ*-environment* $\zeta = (\gamma, \vec{\tau}, \{\preceq_\alpha\}_\alpha)$*, for any* Γ*-*γ*-environment* ρ *and* Γ*-*$\vec{\tau}$*-substitution* $\vec{t}$*, define*

$$\rho \preceq_\zeta^\Gamma \vec{t} \Leftrightarrow \forall x \in \mathrm{dom}(\Gamma),\ \rho(x) \preceq_\zeta^{\Gamma(x)} t_x\,.$$

Then $\rho \preceq_\zeta^\Gamma \vec{t}$ *implies* $[\![s]\!]_{\gamma,\rho} \preceq_\zeta^\sigma s[\vec{\tau}/\Theta][\vec{t}/\Gamma]$.

As usual, the lemma is proved by induction on s.

By the lemma, if $s : \sigma$ then $[\![s]\!] \preceq^\sigma s$. The right-to-left implication of Theorem 6.1.1 follows.

7. Further Operational Properties

The goal of this section is to prove the strictness and operational extensionality theorems claimed in Section 4. For this, we define a second approximation relation between syntax and semantics. If the strictness theorem alone were our goal then it would be possible to simply modify the definitions of Section 6 by systematically replacing strict evaluation with non-strict evaluation.[4] However, in order to prove operational extensionality, we need a more substantial alteration. Our proof adapts the technique of [11], where (in essense) operational extensionality for recursively typed languages is addressed, to the more expressive setting of our polymorphic language.

Definition 7.1 (Non-strict applicative simulation) The relation $\sqsubseteq_{\mathrm{app}}^n$ is the largest relation between closed terms of identical closed type satisfying:

1. if $t \sqsubseteq_{\mathrm{app}}^n t' \colon \sigma \multimap \tau$ then, for all terms $s \colon \sigma$, it holds that $t(s) \sqsubseteq_{\mathrm{app}}^n t'(s) \colon \tau$;
2. if $t \sqsubseteq_{\mathrm{app}}^n t' \colon !\sigma$ then $t \Downarrow^n !\, s$ implies $t' \Downarrow^n !\, s'$ where $s \sqsubseteq_{\mathrm{app}}^n s' \colon \sigma$; and
3. if $t \sqsubseteq_{\mathrm{app}}^n t' \colon \forall\alpha.\,\sigma$ then, for all closed τ, it holds that $t(\tau) \sqsubseteq_{\mathrm{app}}^n t'(\tau) \colon \sigma[\tau/\alpha]$.

[4] To prove adequacy for the non-strict semantics, the quantification over arbitrary terms t in the definition of $\preceq_\zeta^{\sigma_1 \multimap \sigma_2}$ is crucial; cf. Remark 6.3.

For a domain D and closed type σ, a *non-strict approximation relation* between D and σ is a relation $\precsim$ between elements of D and closed terms of type σ satisfying:

(na1) $\{d \mid d \precsim t\}$ is a subdomain of D.
(na2) If $d\!\downarrow$ implies $d \precsim t$ then $d \precsim t$.
(na3) If $d \precsim t$ and $t \sqsubseteq_{\mathrm{app}}^n t'$ then $d \precsim t'$.

We write $\mathcal{A}^n(D,\sigma)$ for the set of all non-strict approximation relations between D and σ.

A *non-strict approximation* Θ*-environment* is a triple $\zeta = (\gamma, \vec{\tau}, \{\precsim_\alpha\}_{\alpha\in\Theta})$, where γ is a Θ-environment, $\vec{\tau}$ is a Θ-substitution and $\precsim_\alpha \ \in \ \mathcal{A}^n(\gamma(\alpha), \tau_\alpha)$. For any type $\sigma(\Theta)$ and non-strict approximation Θ-environment $\zeta = (\gamma, \vec{\tau}, \{\precsim_\alpha\}_\alpha)$, Figure 6 defines a relation $\precsim_\zeta^\sigma \ \in \mathcal{A}^n([\![\sigma]\!]_\gamma, \sigma[\vec{\tau}/\Theta])$. (Again, it takes quite some work to show that the definition is good.)

Lemma 7.2 *Suppose* $\Gamma \mid \delta \ \vdash_\Theta \ s : \sigma$. *For any non-strict approximation* Θ*-environment* $\zeta = (\gamma, \vec{\tau}, \{\precsim_\alpha\}_\alpha)$*, for any* Γ*-*γ*-environment* ρ *and* Γ*-*$\vec{\tau}$*-substitution* $\vec{t}$*, define*

$$\rho \precsim_\zeta^\Gamma \vec{t} \Leftrightarrow \forall x \in \mathrm{dom}(\Gamma),\ \rho(x) \precsim_\zeta^{\Gamma(x)} t_x\,.$$

Then $\rho \precsim_\zeta^\Gamma \vec{t}$ *implies* $[\![s]\!]_{\gamma,\rho} \precsim_\zeta^\sigma s[\vec{\tau}/\Theta][\vec{t}/\Gamma]$.

Again, the proof is by induction on s. The lemma establishes, in particular, that if $s : \sigma$ then $[\![s]\!] \precsim^\sigma s$.

Corollary 7.3 (Non-strict computational adequacy) *If* $t : !\,\sigma$ *then* $t \Downarrow^n$ *if and only if* $\exists d \in [\![\sigma]\!],\ [\![t]\!] = \{d\}$.

Theorem 4.5 (the strictness theorem) is an immediate consequence of the above corollary and Theorem 6.1.1.

Lemma 7.4 *If* $t, t' : \sigma$ *then* $[\![t]\!] \precsim^\sigma t'$ *iff* $t \sqsubseteq_{\mathrm{app}}^n t'$.

Proof (sketch). Using Lemma 7.2, one shows that the relation $[\![t]\!] \precsim^\sigma t'$ satisfies implications 1–3 of Definition 7.1. So $[\![t]\!] \precsim^\sigma t'$ implies $t \sqsubseteq_{\mathrm{app}}^n t'$.

Conversely, if $t \sqsubseteq_{\mathrm{app}}^n t'$ then, by Lemma 7.2, $[\![t]\!] \precsim^\sigma t$. So, by (na3), $[\![t]\!] \precsim^\sigma t'$. □

Proposition 7.5 *If* $t, t' : \sigma$ *then* $t \sqsubseteq_{\mathrm{gnd}} t'$ *iff* $t \sqsubseteq_{\mathrm{app}}^n t'$ *iff* $t \sqsubseteq_{\mathrm{app}}^s t'$.

Proof (sketch). Using Theorem 4.5, it is not hard to show that the relation $t \sqsubseteq_{\mathrm{gnd}} t'$ satisfies implications 1–3 of Definition 4.7. So $t \sqsubseteq_{\mathrm{gnd}} t'$ implies $t \sqsubseteq^{s}_{\mathrm{app}} t'$.

Using the characterization of Proposition 4.8, and Theorem 4.5, one can show that $t \sqsubseteq^{s}_{\mathrm{app}} t'$ satisfies 1–3 of Definition 7.1, so $t \sqsubseteq^{s}_{\mathrm{app}} t'$ implies $t \sqsubseteq^{n}_{\mathrm{app}} t'$.

It remains to show that $t \sqsubseteq^{n}_{\mathrm{app}} t'$ implies $t \sqsubseteq_{\mathrm{gnd}} t'$. Suppose $t \sqsubseteq^{n}_{\mathrm{app}} t'$. By Theorem 4.5, it suffices to show that $x :\_\ \sigma \ \vdash\ C[x] : !\,\tau$ implies $C[t] \sqsubseteq^{n}_{\mathrm{app}} C[t']$. By Lemma 7.4, $[\![t]\!] \precsim^{\sigma} t'$. If $x :\_\ \sigma \vdash C[x] : !\,\tau$ then $[\![C[t]]\!] = [\![C]\!]_{[[\![t]\!]/x]} \precsim^{\sigma} C[t']$, by Lemma 7.2. So, by Lemma 7.4, indeed $C[t] \sqsubseteq^{n}_{\mathrm{app}} C[t]$. □

Theorem 4.9 is an immediate consequence.

8. Justification

In this section, we briefly justify the correctness of our methods for establishing operational properties. All our axioms and arguments can be interpreted within any realizability topos $\mathcal{E}$ satisfying the *strong completeness axiom* of [8], by taking predomains to be the well-complete objects. In *op. cit.* it is explicitly shown that Axioms 1, 2 and 4 of this paper are consequences of strong completeness. Moreover, the validity of Axiom 3 follows from the results of [7]. Thus all results of this paper are true when interpreted within $\mathcal{E}$. It remains to argue that all results with operational content are true in reality.

For this, we observe the following. The evaluation relations $t \Downarrow^{s}$, $t \Downarrow^{s} v$, $t \Downarrow^{n}$ and $t \Downarrow^{n} v$ are all Σ^{0}_{1}. Consequently, the statements $t \sqsubseteq_{\mathrm{gnd}} t'$ and $t \equiv_{\mathrm{gnd}} t'$ are Π^{0}_{2}. Similarly, $t \sqsubseteq^{s}_{\mathrm{app}} t'$ is Π^{0}_{2}, via the characterization of Proposition 4.8. The statement of the strictness theorem (Theorem 4.5) is a bi-implication between Σ^{0}_{1} formulas, and thus also Π^{0}_{2}. The statement of operational extensionality (Theorem 4.9) is a universally quantified bi-implication between Π^{0}_{2} formulas.

It is easily shown that a Π^{0}_{2} sentence holds internally in $\mathcal{E}$ if and only if it is true in reality (for the right-to-left implication, an unbounded search provides the realizer). It follows that implications between Π^{0}_{2} statements that are valid in $\mathcal{E}$ are also true in reality. Therefore, any operational conclusion of one of the forms discussed above, obtained as a result of the methods of this paper, is indeed true.

References

[1] R. Amadio. On the adequacy of PER models. In *Mathematical Foundations of Comp. Sci. 1993*, pages 222–231. Springer LNCS 711, 1993.

[2] G. M. Bierman, A. M. Pitts, and C. V. Russo. Operational properties of Lily, a polymorphic linear lambda calculus with recursion. *Elect. Notes in Theor. Comp. Sci.*, 41, 2000.

[3] P. J. Freyd, E. P. Robinson, and G. Rosolini. Functorial Parametricity. In *Proc. 7th Symposium on Logic in Comp. Sci.*, pages 444–452, Santa Cruz, 1992.

[4] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

[5] J. M. E. Hyland. A small complete category. *Annals of Pure and Applied Logic*, 40:135 – 165, 1988.

[6] J. M. E. Hyland. First steps in synthetic domain theory. In *Category Theory, Proc. Como 1990*, pages 131–156. Springer LNM 1488, 1991.

[7] J. M. E. Hyland, E. P. Robinson, and G. Rosolini. The discrete objects in the effective topos. *Proc. Lond. Math. Soc.*, 3(60), 1990.

[8] J. R. Longley and A. K. Simpson. A uniform account of domain theory in realizability models. *Math. Struct. in Comp. Sci.*, 7:469–505, 1997.

[9] W. K.-S. Phoa. Effective domains and intrinsic structure. In *Proc. 5th Annual Symposium on Logic in Comp. Sci.*, pages 366–377, 1990.

[10] W. K.-S. Phoa. Two reults on set-theoretic polymorphism. In *Category Theory in Comp. Sci.*, pages 219–235. Springer LNCS 530, 1991.

[11] A. M. Pitts. A note on logical relations between semantics and syntax. *Logic Journal of the Interest Group in Pure and Applied Logics*, 5(4):589–601, 1997.

[12] G. D. Plotkin. Type theory and recursion. Invited talk at *8th Symposium on Logic in Comp. Sci.*, 1993.

[13] G. D. Plotkin and A. J. Power. Computational effects and operations: an overview. *Elect. Notes in Theor. Comp. Sci.*, to appear, 2004.

[14] B. Reus and T. Streicher. General synthetic domain theory — a logical approach. *Math. Struct. in Comp. Sci.*, 9:177–223, 1999.

[15] J. C. Reynolds. Types, abstraction and parametric polymorphism. In *Information Processing '83*, pages 513–523. North Holland, 1983.

[16] E. P. Robinson. How complete is PER? In *Proc. 4th Symposium on Logic in Comp. Sci.*, pages 106–111, 1989.

[17] E. P. Robinson and G. Rosolini. Reflexive graphs and parametric polymorphism. In *Proc. 9th Symposium on Logic in Comp. Sci.*, pages 364–371, 1994.

[18] G. Rosolini. *Continuity and Effectivity in Topoi*. PhD thesis, University of Oxford, 1986.

[19] A. Ščedrov. Intuitionistic set theory. In *Harvey Friedman's Research on The Foundations of Mathematics*, pages 257–284. Elsevier Science Publishers, 1985.

[20] D. S. Scott. Relating theories of the λ-calculus. In J. Seldin and J. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 403–450. Academic Press, 1980.

[21] A. K. Simpson. Computational adequacy in an elementary topos. In *Comp. Sci. Logic, Proc. CSL '98*, pages 323–342. Springer LNCS 1584, 1999.

[22] A. K. Simpson. Computational adequacy for recursive types in models of intuitionistic set theory. In *Proc. 17th Symposium on Logic in Comp. Sci.*, pages 282–298, 2002.

[23] A. K. Simpson and G. D. Plotkin. Complete axioms for categorical fixed-point operators. In *Proc. 15th Symposium on Logic in Comp. Sci.*, pages 30–41, 2000.